Going Native:

Indexing architecture of eXist-db_ An Open Source native XML database system

Raghad Yaseen Lazim[1]
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang_UMP
Pahang, Malaysia
Raghad.altaai@yahoo.com

Adzhar Bin Kamaludin[2]
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang_UMP
Pahang, Malaysia
adzhar@ump.edu.my

***Abstract*—XML is a markup language specifically designed for the Internet, and it has become the standard for represent and exchange the data on the Internet due to the ease of processing the exchanged information. However, the challenge is to serve and access XML data within the database efficiently. The difference between the nature of the database structure and XML data structure became an important research topic, and this led to emergence eXist-db, a native XML database which designed specifically to store large sizes of XML documents. This paper presents the indexing architecture of eXist-db, a native XML database and modified the query engine of eXist by indexing schema supports quick identification of structural node relationships thereby enhances query process.**

Keywords—eXist; Native XML database; XML; Query

I. INTRODUCTION

With the development and popularization of the Web technologies, as well as the XML data as a standard to present and exchange data on the internet, significant amount of XML documents are being added to the network continuously in various application domains. As a result, storing, indexing and querying large collection of XML documents have become an important issue [1][2]. The difference between the nature of the database structure and XML data structure, leads to some problems and important issues such as: Involved in storing data, retrieval from the database, and Lack of direct accessible in the database. Thus, we have a problem of designing the database, and this defect in databases prevents the maximum utilization of flexible XML which leads to problems in performance [3]. Current research focuses more and more on how to manage large amount of XML data effectively [4][5]. The advantages of the Native storage has been widely accepted, designed for processing XML data in native XML databases have received increasing attention, and has a very bright prospect [6].

An indexing scheme enhances query process and supports quick identification of structural node relationships. By indexing schema, can extend the application of path join algorithms to implement most parts of the XPath query language specification and add support for keyword search on element and attribute contents [7][8].

In this paper we introduce eXist_db's features and provide a quick overview of how to develop standalone and Web-based applications using the different interfaces provided with eXist. Finally, we will take a closer look at eXist's indexing and storage architecture to see how query processing works.

II. OVERVIEW

1. eXist Background

eXist database is the most popular open source management system built using XML technology. eXist cover most of the basic features of the Native XML database as well as a number of advanced technologies such as keyword searches on text, queries on the proximity of terms, and regular expression-based search patterns. In addition, it is very good in the query response time, very easy to use, runs on most platforms, and it's easy to set up and operate, because of it is platform independent [9][1].

eXist_db in hierarchical collections provides schema-less storage of XML documents. Using an extended XPath syntax [10][11], users may query a distinct part of the collection hierarchy or even all the documents contained in the database. Despite being lightweight, eXist's query engine implements efficient, index-based query processing.

2. eXist-db architecture

The overview of eXist-db system architecture is shown in Figure 1. Be dealt with all calls backend storage by broker class, implementing the database broker. Applications may access a remote eXist server via the interfaces such as: XMLRPC XML, SOAP, Rest, and WebDAV. API interface supported by eXist, which supports the embedding the database into an application without running an external server.

Currently, eXist's XML RPC interface makes multiple RPC calls for retrieving the the XQuery results. The first RPC just gets the ids of the XML document and the positions of the nodes to be displayed within each document. Then a loop is run where in each iteration an RPC call is made by passing the XML document id and the position of the node id to get the actual XML node content. This was modified to have a single RPC call to gets all the required nodes of the XML document as a string. This enabled collation of results from various distributed servers without congesting the network. The XPath

engine is used to parse the XPath to find the clusters which are to be queried [12][13].

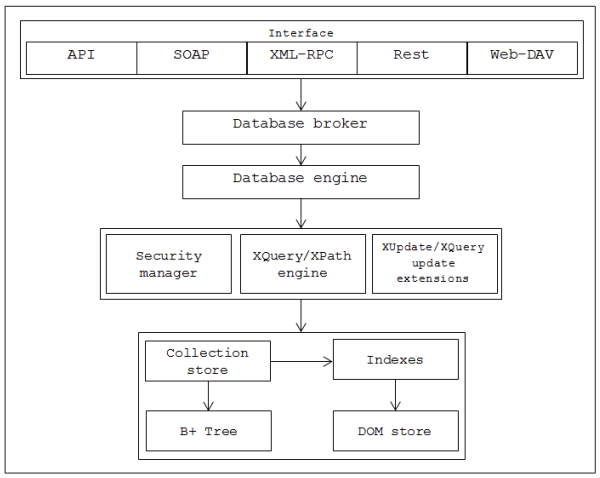

Figure 1. eXist-db system architecture

III. EXIST-DB INDEXING

Database indexes are used extensively by eXist-db to facilitate efficient querying of the database. eXist-db has been designed to provide XPath queries and its own XQuery implementation, which cannot understood without knowing the indexing schema by using indexes for all element, text, and attribute nodes. An indexing scheme in the database supports quick identification of structural relationships between nodes, such as parent-child (P-C), ancestor-descendant (A-D) [17]. Based on path join algorithms, a wide range of query expressions are processed using only index information [7][8].

eXist-db widely used database indexes to facilitate efficient querying of the database. Figure 2.show indexing system architecture.

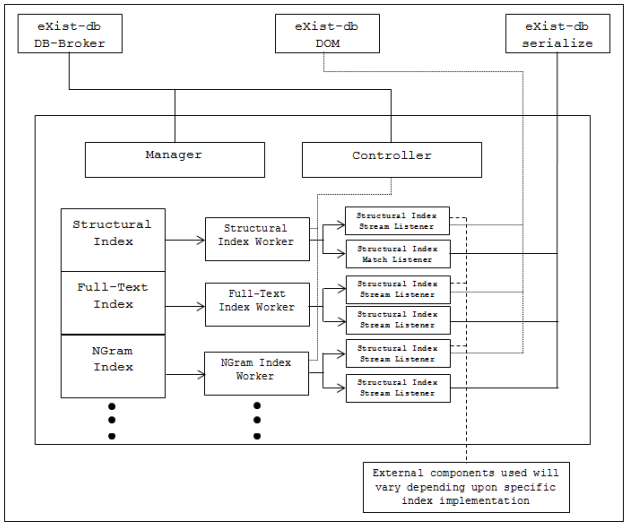

Figure 2. Indexing system architecture [15]

The new version of eXist-db 1.2, features a new indexing architecture. Where the indexes have been moved from the database core to be as pluggable extensions:

1. *Structural Indexes*

Structural indexing is a check for understanding that utilizes key concept words. These index the nodal structure, elements (tags) and attributes, of the documents in a collection as a basis for content writing. It is created and maintained automatically in eXist, and used by all non-wildcard XPath and XQuery expressions in eXist (not "//*").Structural indexes have also been used as statistical synopses for estimating selectivity's of path expressions. This index is stored in the database file elements.dbx [16][16].

For structural indexes to be practical, the index needs to map all the elements and attributes to help the query engine to resolve queries for a given XPath expression. For example:

//book/section $\alpha + \beta$

The basic approaches for eXist indexing for querying is:

- Lookup for <book> node and then <section> node. Whereas: <book> node is parent for <section> node
- Compute the structural relationship between the set's nodes, such as: Parent–Child (P-C), Ancestor–Descendant (A-D), or mixed types of both relationships, in an XML database [17][18]

2. *Range Indexes*

Range indexes are special access methods used to retrieve data within the eXist-db. This index allows users to select nodes by identifying a single value or range of values, which must match or contain at least one value contained within an identified field by way of standard XPath operators and functions.

```
<books>
  <book ISBN="94587403">
    <title> Beginning XML</title>
    <price>65.76</price>
  </book>
  <book ISBN="70156878">
    <title>XML in Technical Communication</title>
    <price>81.90</price>
  </book>
</books>
```

The node <price> has value as floating-point number (for example: "65.76"), this value has XSD data type of XS: double. eXist-db during the indexing will apply the data type for

<price> values as double floating point numbers, and ignore all the values are not as double floating point numbers. Then, range index will use expression to compare <price> with XS: double value. For example:

//book[price > 135.0]

Instead of retrieval each element value and check it if XS: double or not, range index provide the query engine with a more efficient method of data conversion. Without range indexes, eXist-db has to do full scan for the <price> elements, and this consumes a long time especially with the large amount of data [19].

3. Spatial Indexes and NGram index

eXist-db currently came with two index types, to provide a new method to index XML data. These indexes are:

3.1 Spatial index is a type of extended index that listens for spatial geometries described through the Geography Markup Language (GML) in the source XML to enable spatial queries. A spatial index will store some of the geometric characteristics of Geography Markup Language geometries [20].

3.2 NGram indexes

An NGram index is a contiguous sequence of N-characters. It is a powerful and useful method for substring searching, queries on scripts and languages such as Chinese and Japanese which are without word breaks. An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (N – 1) order Markov model [21]. The following table shows 3-grams example:

TABLE I. THE CORRESPONDING 1-GRAM, 2-GRAM AND 3-GRAM SEQUENCES

Unit	Sample sequence	1-gram sequence	2-gram sequence	3-gram sequence
		unigram	bigram	trigram
Word	…Hi there…	…,Hi, there,…	…,Hi there,…	Hi there_
character	…Hi _there…	…,H,i,_,t,h,e,r,e,…	…,H,i,i_,_t,th,he,er,re,e_,..	…,Hi_,i_t,_th,he_,e_r,_re,…

4. Optional Indexes

In the new indexing architecture for eXist-db, some indexes have been moved from the database core. Legacy Full Text, The full text, and XML-id are optional indexes.

Legacy Full Text Indexes: This index is used to query for a sequence of separate "words" or tokens in a longer stream of text. While building the index, the text is parsed into single tokens which are then stored in the index.

Full Text Indexes: This index refers to the technology that allowing you to efficiently searching full text or single document within the database. Full text index depends on metadata or on parts of the original texts represented in databases (such as titles, abstracts, selected sections, or bibliographical references) [16].

XML-ID Index: An index of all xml:id attribute values is automatically created. These values can be queried by fn:id().

IV. INDEXES AND RE-INDEXING

The establishment and management of indexes is a common eXist-db system task. So you do not need to update an index when you update a database document or collection, because eXist-db will update the indexes via XUpdate or XQuery Update expressions. eXist-db provides a mechanism to create and configure indexes through a XML configuration file. This configuration file allows to re-creation of same indexes in the startup time.

After create the indexes, eXist-db will automatically updates and maintains indexes defined by the user (administrator). Thus, the user will not update the index when update the database. When you add new index definition to your database collection, eXist-db will apply this new index to a new data added to the collection (or its sub-collections). RE-INDEXING is used to apply the new indexes settings on all the existing collections [22].

V. PERFORMANCE AND CONCLUSION

eXist-db use indexes to improve query processing performance on XML data. The indexing ability of eXist-db allows fast access to data by using different structural representations to reference the data [14].

Full-text indexing system makes it easy to create high-performance document search systems with eXist, which can dramatically increase the speed and sophistication of full text searches. Range or NGram indexes improve the performance of queries that depend on string or value comparisons. Exist the some indexes in the cache memory, reduces I/O operations and the need to access the raw DOM to complete a query, because it's allow eXist-db to load the right amount of data in the memory.

REFERENCES

[1] J. Zhang, B. Lang and Y. Duan, "An XML Data Placement Strategy for Distributed XML Storage and Parallel Query", IEEE, 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (2012), pp. 433 - 439.

[2] Y. Yang, L. Hai-ge, "Querying XML Data Based on Improved Prefix Encoding", IEEE, International Conference on Computer Application and System Modeling (ICCASM 2010), vol.10, pp. 521-525.

[3] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohmann, "On Supporting Containment Queries in Relational Database Management Systems", In Proceedings of the SIGMOD Conference, 2001, Santa Barbara, California, USA.

[4] Y. Yang, "benchmarking of Native XML database systems", MSc. Univ. of Wollongong, school of information technology and computerscience, Wollongong, 2005.

[5] S.Haw and C. Lee, "Data Storage Practices and Query Processing in XML Databases" A Survey. J. Knowledge-Based Systems. vol 24, (2011), pp. 1317–1340.

[6] H.M. Khorasani, "PERFORMANCE ANALYSIS OF XQUERY VS. SQL", MSc. of Computer Science, Univ. of Wisconsin Platteville, Univ.of Applied Sciences Darmstadt, 2004.

[7] O. Logvynovskiy and K. Lu, "Structural Join Algorithm for Sequential Regular Path Expressions", Journal of Computing and Information Technology-CIT 12, 2004, vol. 3, pp. 237–250.

[8] D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, and Y Wu, "Structural joins: A primitive for efficient xml query pattern matching:, Conference on Data Engineering (18th), San Jose, California, 2002.

[9] Divesh Srivastava, Shurug Al-Khalifa, H.V. Jagadish, Nick Koudas, Jignesh M. Patel, and Yuqing Wu. “Structural Joins: A Primitive for Efficient XML Query Pattern Matching”. In Proceedings of the ICDE Conference, 2002.

[10] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. “XML Path Language (XPath) 2.0", W3C Working Draft 30 April 2002. http://www.w3.org/TR/xpath20.

[11] J. Clark and S. DeRose, “XML Path Language (XPath) Version 1.0”, W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath. W3C Recommendation.

[12] A.M. Kulkarni, D. Shah, J. Thirunavukkarasu, P.S. Pillai, S. Sulegai, "Distributed eXist-db", Technical Report IIITB-OS-2011-10A, International Institute of Information Technology, Bangalore.

[13] B. Chaudhri , A. Rashid (Author), R. Zicari, "XML Data Management: Native XML and XML-Enabled Database Systems", Addison-Wesley Professional, 1 edition 2003, pp. 688.

[14] L. Cahlander, P. Cross, Z. Geldiyev, A. Stallman and M. Turpin, "EXIST-DB OPEN SOURCE XML DATABASE SOFTWARE ARCHITECTURE DESCRIPTION", W3C: Introduction to Software Architecture, 2010.

[15] J. Cheng and W. Ng, "XQzip: Querying Compressed XML Using Structural Indexing", https://www.cse.ust.hk/~wilfred/paper/edbt04.pdf.

[16] http://exist-db.org/exist/apps/doc/indexing.xml#D2.2.2

[17] S-C.Haw and C-S. Lee, "Data storage practices and query processing in XML databases: A survey", Knowledge-Based Systems, 2011, vol.24, pp. 1317–1340.

[18] Y. Xin, Z. He and J. Cao, "Effective pruning for XML structural match queries", Data & Knowledge Engineering, 2010, vol. 69, pp. 640–659

[19] Range Indexing, Vers. 4.0, KE Software Pty Ltd, 2009, http://www.kesoftware.com/downloads/EMu/documents/Range%20Indexing.pdf

[20] http://exist-db.org/exist/apps/doc/devguide_indexes.xml#spatial

[21] http://exist-db.org/exist/apps/doc/ngram.xml

[22] Q. Li and B. Moon, “Indexing and Querying XML Data for Regular Path Expressions”, In VLDB 2001, Proceedings of 27th International Conference on Very Large Databases, September 11-14, 2001, Roma, Italy.